

Lecture Topics

- an overview of common approaches
 - dimensions for parallel models (wrap-up)
 - more complex approaches

Administrivia

- [none]

- (last dimension...hardware)
- and a hardware dimension that interacts with sharing control:
 - shared memory: processors share a common pool of memory
 - distributed memory: processors have private pools of memory
 - many variations/combinations, but let's start simple
 - software can be used to masquerade here, too

More Complex Models of Parallelism

- hidden ISA/compiler parallelism (ILP, VLIW, EPIC)
 - (the most common form of parallelism!)
 - one thread
 - parallelism fully implicit (explicit after compiler in VLIW/EPIC)
 - sharing asynchronous
 - data have no owner
 - but not important with implicit parallelism
- exposed ISA parallelism
 - single-instruction, multiple data (SIMD): MMX, SSE, vectors
 - again, one thread (parallel hardware within one processor)
 - parallelism
 - sometimes implicitly extracted from loops
 - otherwise hand-coded in assembly
 - as above: sharing asynchronous

- data parallel loops
 - loop body parallel by analysis and/or programmer assertion
 - from language perspective, one thread
 - runtime implementation may use
 - one thread and vector ISA
(vector supercomputers, some MPPs, Sony Playstation 2?, etc.)
 - dynamic number of threads that exist only during loop
(some OpenMP implementations, albeit using a thread pool)
 - one thread per available processor
(data parallel language on distributed memory hardware)
 - parallelism mostly implicit
 - except for assertion of independence
 - possibly thread-count annotations
 - sharing technically depends on target hardware platform,
but no shared accesses in loop body!

- shared memory threads (e.g., Posix)
 - dominate the server market
 - sometimes encapsulated in language (e.g., Java)
 - dynamic number of threads
 - sharing an address space
 - thread to processor mapping varies over time
 - explicit parallelism: each thread has a main function
 - asynchronous sharing: shared memory machines only

- message passing
 - e.g., MPI, the most widely used API for high-performance computing
 - (Bill Gropp, who with Rusty Lusk standardized MPI and provided an early open implementation, is now a CS faculty member here)
 - fixed number of threads (one per processor)
 - explicit parallelism (SPMD)
 - synchronous sharing
 - data are owned by a thread
 - data must be explicitly sent from one thread to another
 - MPI 2.0 incorporates “one-sided” communication
 - allows a thread to name data owned by another processor
 - can be used asynchronously (choice is programmer’s)
- Unified Parallel C (UPC): “A descendant of Split-C, AC, and PCP, ...”
 - simple and efficient; used widely by three letter agencies
 - shared/private data qualifiers on variables
 - global address space with put/get capability
 - fixed number of threads (one per processor)
 - explicit parallelism (SPMD)
 - synch/asynch: programmer-controlled, but performance is heavily implementation/platform dependent
 - Split-C is synchronous!

- openMP (MP = multiprocess...)
 - combination of data parallelism and shared memory threads
 - viewed for some time as competitor to MPI
 - particularly for large shared memory systems
 - which have essentially vanished
 - OpenMP requires shared memory
 - hybrids with MPI still sometimes used (clusters of SMP/multi-core)
 - dynamic number of threads
 - parallel regions
 - new thread creation
 - thread joining (or waiting)
 - explicit parallelism (several levels possible, chosen by programmer)
 - asynchronous sharing

- actors/data-driven execution/compiler threads/etc.
 - old model, possibly seeing resurgence in some server applications
 - also has nice properties for HPC
 - and some major success stories
 - as well as a couple of drawbacks
 - I'm going to simplify slightly...
 - thread-to-data binding is strong
 - object (“actor”) receives message
 - computes, sends messages to others, etc.
 - goes to sleep until next message arrives
 - dynamic number of threads (one per object)
 - explicit parallelism
 - essentially separate binaries
 - although can technically share code
 - synchronous sharing: all messages to one actor are serialized